Labwork: Working with algebraic types

December 6, 2018

1. Consider the `Tree a` type for binary trees, defined by

   ```
   data Tree a = Nil | Node a (Tree a) (Tree a)
                 deriving (Eq,Ord,Show,Read)
   ```

   Define the following operations:

   (a) `collapse::Tree a ->[a]` which returns the list of values in the nodes of a tree, enumerated by an inorder traversal.

   (b) `size::Tree a->Int` which returns the number of nodes in the tree.

   (c) `mapTree::(a->b) -> Tree a -> Tree b` such that the function call `(maptree f t)` returns the binary tree obtained from t by replacing the content v of every node with `f` v.

   (d) `sumTree::(Num a)=>Tree a -> a` which computes the sum of values stores in the interior nodes of a binary tree.
   (Note that `(sumTree Nil)` should be 0)

2. Define an algebraic type `Student` to record the name (a `[Char]`) and grade (a `Float`) at *Advanced Logic and Functional Programming* (ALFP) for every student. Afterwards, define the following operations:

   (a) `bestStudents::[Student]->[[Char]]` which takes as input a list of students, and returns the list of names of the student with best grade.

   (b) `meanGrade::[Student]->Float` which takes as input a nonempty list of students and computes their average grade at ALFP.

3. Consider the algebraic type `(Expr a)` to represent arithmetic expressions for numeric types:

```
data Expr a = Val a
            | Var [Char]
            | Sum (Expr a) (Expr a)
            | Prod (Expr a) (Expr a)
            | Minus (Expr a) (Expr a)
```

Define the operation

```
compute :: Num a => [([Char],a)] -> Expr a -> a
```

which takes as inputs

(a) a list of pairs that indicate the value of every variable, and

(b) an arithmetic expression `expr`

and computes the value of `expr`.

For example:

```
compute [("x",1),("y",7)]
        (Sum (Prod (Var "x") (Val 5))
             (Minus (Val 9) (Var "y")))
```

should compute the value of $\mathtt{x} \cdot 5 + (9 - \mathtt{y})$, which is 7.